UNIVERSIDADE
DE ÉVORA

Relatório do trabalho prático do

JOGO DA MOEDA

Realizado por:

- Diogo Marona – l63164
- João Padeiro – l63655
- Tomás Rato – l62755

UNIVERSIDADE
DE ÉVORA

Índice

1. Introdução

O presente trabalho tem como objetivo o desenvolvimento de um jogo de estratégia denominado Jogo das Moedas, implementado na linguagem de programação C. Este jogo, apesar de possuir regras simples, apresenta uma profundidade estratégica que desafia as capacidades lógicas e analíticas dos jogadores.

O objetivo principal deste trabalho é aplicar os conceitos de programação aprendidos em sala de aula para criar um software funcional. Além disso, busca-se desenvolver a capacidade de resolução de problemas e trabalho em equipa, ao implementar um jogo interativo com regras bem definidas, modos de jogo variados e funcionalidades adicionais como salvar e carregar partidas.

2. Decisões técnicas tomadas

A implementação do Jogo das Moedas foi planeada de forma modular, com o objetivo de garantir clareza, reutilização de código e facilidade de manutenção. Abaixo, detalham-se as principais decisões técnicas adotadas:

2.1. Estruturação do Código

O programa foi dividido em funções bem definidas, cada uma responsável por uma tarefa específica, como por exemplo, cálculo de moedas restantes e manipulação de ficheiros. Essa abordagem modular permite isolar responsabilidades, facilitando a leitura do código.

2.2. Estruturas de Dados

Foi utilizada uma struct chamada **Fila** para representar cada fila de moedas. Essa abordagem permitiu encapsular informações relacionadas a cada fila e facilita a manipulação de dados durante o jogo. A alocação dinâmica de memória foi utilizada para criar o número de filas definido pelo usuário, promovendo maior flexibilidade.

2.3. Manipulação de ficheiros

Para possibilitar a funcionalidade de salvar e retomar jogos, o estado do jogo é armazenado em ficheiros. São gravados dados como o número de filas, a quantidade de moedas em cada fila e a vez do jogador. O programa verifica se o ficheiro existe antes de carregar um jogo, garantindo robustez.

2.4. Validação de Dados

O programa implementa validação rigorosa das entradas do usuário, garantindo que apenas números inteiros positivos sejam aceitos para a configuração inicial e para as jogadas. Isso previne comportamentos inesperados e erros, e mantém a consistência das regras do jogo.

2.5. Interatividade

O menu interativo foi projetado para guiar o usuário de forma intuitiva, apresentando opções claras para iniciar novos jogos, carregar jogos salvos ou encerrar o programa. Mensagens de erro informativas são exibidas em caso de entradas inválidas.

2.6. Gestão de Memória

Funções como malloc e free foram usadas para gerir a memória alocada dinamicamente. O programa garante a liberação de toda a memória utilizada antes de terminar a execução, evitando fugas de memória (memory leaks).

3. Algoritmos implementados

A lógica do Jogo das Moedas foi baseada em algoritmos que atendem aos diferentes modos de jogo. Cada modo foi projetado para oferecer uma experiência específica, desde o confronto direto entre dois jogadores até à interação com o computador, utilizando estratégias básicas ou avançadas.

3.1. Algoritmo do Modo Humano X Humano

Neste modo, dois jogadores alternam turnos para realizar as suas jogadas. O programa verifica:

- A validade da jogada (se a fila e a quantidade de moedas escolhidas são válidas).
- A atualização do estado do jogo, exibindo o número de moedas restantes após cada jogada.
- A condição de vitória ou derrota, encerrando o jogo quando apenas uma ou nenhuma moeda permanece.

3.2. Algoritmo do Modo Humano X Máquina Básica

A máquina realiza jogadas aleatórias, baseando-se nas seguintes etapas:

- Selecionar uma fila que ainda contenha moedas.
- Escolher um número aleatório de moedas para retirar dessa fila.
- Atualizar o estado do jogo e verificar as condições de vitória ou derrota.

3.3. Algoritmo do Modo Humano X Máquina Avançada

Este modo utiliza técnicas mais elaboradas para a tomada de decisão da máquina. A estratégia avançada envolve:

- Análise do número de filas e moedas restantes para identificar jogadas vantajosas.
- Priorização de manter o adversário em situações desfavoráveis, como filas com poucas moedas.

- Jogadas calculadas para criar cenários de vitória inevitável, sempre que possível.
- Apresenta resposta para qualquer situação de jogo possível com o objetivo (anteriormente mencionado) de criar cenários de vitória inevitável.

Após pesquisa, reparámos que os métodos matemáticos usados para colocar a máquina numa posição vencedora se baseiam na utilização de XOR's (operação lógica de OR exclusivo), método tal que não conseguimos entender com clareza. Logo, depois do grupo se aprofundar nesta técnica optámos por um método comparável. Então, este algoritmo foi obtido com diversos testes e correção de erros até chegarmos ao algoritmo apresentado no código.

3.4. Gestão do Estado do Jogo

Independente do modo escolhido, o programa:

- Mantém o estado atualizado das filas e moedas em memória.
- Permite salvar e carregar o estado do jogo de ficheiro para continuação posterior.
- Apresenta mensagens claras para guiar o usuário durante a partida.

3.5. Criação de tabuleiro

As funções de criação de tabuleiro são fundamentais para garantir a fácil leitura e otimização do código:

- **DefinirFilas**: Define as filas do tabuleiro e garante que o número de filas é um inteiro positivo antes de iniciar o jogo.
- **DefinirMoedas**: Define as moedas por fila e valida que a quantidade de moedas para cada fila seja maior que zero e um número inteiro.
- Estas funções utilizam loops para repetir a solicitação de entrada até que valores válidos sejam inseridos pelo usuário.

3.6. Cálculo de moedas restantes

A função **TotalMoedas** soma todas as moedas restantes nas filas para:

- Determinar se o jogo continua ou se há um vencedor.
- Evitar a execução de jogadas desnecessárias quando o jogo já estiver decidido.

3.7. Tratamento de ficheiros

Para salvar e carregar o estado do jogo, as seguintes práticas foram adotadas:

- Verificação da existência do ficheiro antes de carregar o jogo.
- Utilização de formatação consistente para gravar e ler dados (ex.: número de filas, moedas por fila, e jogador ativo).
- Exclusão automática do ficheiro de salvamento ao final de uma partida que tenha sido carregada a partir de um ficheiro.

Independente do modo escolhido, o programa:

- Mantém o estado atualizado das filas e moedas em memória.
- Permite salvar e carregar o estado do jogo de ficheiros para continuação posterior.
- Apresenta mensagens claras para guiar o usuário durante a partida.

3.8. Estrutura do código restante

A função **main** é o ponto de entrada do programa e foi projetada para gerir o fluxo principal do jogo. As suas responsabilidades incluem:

- Apresentar o menu inicial interativo, permitindo ao usuário escolher entre iniciar um novo jogo, carregar um jogo salvo ou sair do programa.
- Gerir a seleção de modos de jogo e iniciar os respetivos fluxos, como "Humano X Humano" ou "Humano X Máquina".
- Integrar as funcionalidades de validação, manipulação de ficheiros e gestão de memória, garantindo que cada etapa do jogo seja executada de forma ordenada e segura.

- Oferecer robustez ao tratar entradas inválidas do usuário e assegurar que o programa encerre adequadamente, liberando todos os recursos alocados.

Todos os aspetos necessários foram abordados de forma a manter o código seguro e sem erros.

4. Instruções para compilação e execução

4.1. Compilação do programa

As etapas seguintes indicam como compilar o código:

1. Certifique-se de que o código-fonte do programa (neste caso o "jogodamoeda.c") esteja localizado na diretória atual. Caso não esteja pode utilizar o comando *cd <diretória>* para aceder à diretória desejada, ou então colocar o ficheiro na pasta onde o terminal será aberto.
2. Abra um terminal ou prompt de comando.
3. Execute o comando a seguir para compilar o código:

   gcc -o jogodamoeda jogodamoeda.c

O compilador criará um ficheiro executável chamado *jogodamoeda*.

4.2. Execução do programa

Após a compilação, o programa pode ser executado diretamente no terminal:

1. No terminal, execute o comando para iniciar o programa:

   ./jogodamoeda
2. O programa exibirá o menu inicial com as opções disponíveis. Escolha a opção desejada inserindo o número correspondente.

4.3. Navegação pelos Menus

- Opção 1: *Iniciar Novo Jogo*:
  Pode escolher o modo de jogo que pretende:
  - *1*: Humano X Humano
  - *2*: Humano X Máquina Básica
  - *3*: Humano X Máquina Avançada

  Configure o número de filas e moedas, conforme solicitado.
- Opção 2: *Continuar Jogo Guardado*:
  - Se houver algum jogo previamente salvo, ele será carregado automaticamente e retomando do ponto onde foi interrompido.
  - Caso não exista um jogo salvo, o programa exibirá uma mensagem de erro.
- Opção 3: *Sair*:
  - Fecha o programa.

4.4. Salvar e retomar jogos

Durante o jogo, é possível salvar o estado atual inserindo *0 0* como entrada ao ser solicitado o número de moedas e a fila (durante um jogo). O jogo será salvo no ficheiro *jogoguardado.txt* e poderá ser retomado posteriormente escolhendo a opção *2* no menu inicial.

4.5. Erros e validação de entrada

O programa valida as entradas do usuário para evitar erros. Por exemplo:

- Números negativos ou não inteiros para definir filas não são aceites.
- Caso uma entrada inválida seja detetada, o programa solicitará ao usuário que insira um valor válido.

Para realizar uma jogada válida deverá apresentar o número correspondente da fila que quer tirar moedas e a quantidade de moedas que pretende retirar da mesma. Por exemplo, o comando *3 2* irá retirar duas moedas da fila 3.

4.6. Requisitos do sistema

É necessário:

- Sistema operativo com suporte ao GCC (Linux, Windows via MinGW, ou macOS).
- Compilador GCC ou compatível.

A biblioteca utilizada foi apenas a biblioteca padrão C (*stdio.h*, *stdlib.h*, etc).

5. Testes e resultados

5.1. Cenários de teste

Os testes foram realizados para validar a funcionalidade em diferentes cenários, conforme descrito no enunciado do trabalho. Os principais cenários testados incluem:

1. Configuração inicial do jogo:
    - Entrada válida:
      Entrada de valores positivos inteiros para o número de filas e de moedas em cada fila. Exemplo:

```
Quantas filas deseja no seu jogo?
3
Quantas moedas tem a fila 1?
5
Quantas moedas tem a fila 2?
8
Quantas moedas tem a fila 3?
3
```

    - Entrada inválida:
      Exemplos:

```
Quantas filas deseja no seu jogo?
-1
Digite um número inteiro positivo!
Quantas filas deseja no seu jogo?
1.6
Digite um número inteiro positivo!
Quantas filas deseja no seu jogo?
0
Digite um número inteiro positivo!
Quantas filas deseja no seu jogo?
asdas
Digite um número inteiro positivo!
Quantas filas deseja no seu jogo?
3
```

2. Jogadas no Modo Humano X Humano:
    - Entrada válida:
      Exemplos:

```
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
2 3
Estado do jogo:
Fila 1: 5 moedas
Fila 2: 5 moedas
Fila 3: 3 moedas
Jogador 2 - Escolha a fila de moedas e o número de moedas a retirar.
3
3
Estado do jogo:
Fila 1: 5 moedas
Fila 2: 5 moedas
Fila 3: 0 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
```

    - Entrada inválida:
      Exemplos:

```
Jogador 2 - Escolha a fila de moedas e o número de moedas a retirar.
1 5
Digite valores possíveis para este jogo!
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 4 moedas
Fila 3: 5 moedas
Jogador 2 - Escolha a fila de moedas e o número de moedas a retirar.
1 -1
Digite valores possíveis para este jogo!
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 4 moedas
Fila 3: 5 moedas
Jogador 2 - Escolha a fila de moedas e o número de moedas a retirar.
as as
Digite valores possíveis para este jogo!
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 4 moedas
Fila 3: 5 moedas
Jogador 2 - Escolha a fila de moedas e o número de moedas a retirar.
```

3. Jogadas no Modo Humano X Máquina (Básico e Avançado):
    - Máquina Básica:
      A máquina faz escolhas aleatórias válidas. Exemplo de interação:

```
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
1 4
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 5 moedas
Fila 3: 6 moedas
A máquina escolheu a fila 2 e retirou 1 moeda(s).
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 4 moedas
Fila 3: 6 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
3 5
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 4 moedas
Fila 3: 1 moedas
A máquina escolheu a fila 3 e retirou 1 moeda(s).
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 4 moedas
Fila 3: 0 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
```

    - Máquina Avançada:
      A máquina utiliza estratégias par maximizar as suas chances de vitória. Exemplos de interação:

```
Estado do jogo:
Fila 1: 2 moedas
Fila 2: 5 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
2 3
Estado do jogo:
Fila 1: 2 moedas
Fila 2: 2 moedas
A máquina escolheu a fila 1 e retirou 2 moeda(s).
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 2 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
```

4. Salvar e continuar o jogo:
    - Durante o jogo, inserir *0 0* salva o estado do jogo, independentemente do modo de jogo. Exemplo:

```
Jogo Iniciou!
Para guardar o jogo, no estado atual, inserir 0 0.
Estado do jogo:
Fila 1: 5 moedas
Fila 2: 4 moedas
Fila 3: 6 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
1 4
Estado do jogo:
Fila 1: 1 moedas
Fila 2: 4 moedas
Fila 3: 6 moedas
A máquina escolheu a fila 2 e retirou 3 moeda(s).
Estado do jogo:
Fila 1: 1 moedas
Fila 2: 1 moedas
Fila 3: 6 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
0 0
O jogo foi guardado.
Bem-vindo ao Jogo da Moeda!
Por favor digite a opção que pretende:
    1 - Iniciar Novo Jogo
    2 - Continuar Jogo Guardado
    3 - Sair
```

    - Ao escolher a opção 2 no menu inicial, o jogo guardado é retomado. Exemplo:

```
Bem-vindo ao Jogo da Moeda!
Por favor digite a opção que pretende:
    1 - Iniciar Novo Jogo
    2 - Continuar Jogo Guardado
    3 - Sair
2
Continuar Jogo!
Estado do jogo:
Fila 1: 1 moedas
Fila 2: 1 moedas
Fila 3: 6 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
```

5. Fim do jogo:
    - Quando restar apenas uma moeda ou nenhuma, o algoritmo procede a finalizar o jogo. Exemplos:

```
Jogo Iniciou!
Para guardar o jogo, no estado atual, inserir 0 0.
Estado do jogo:
Fila 1: 2 moedas
Fila 2: 1 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
1 2
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 1 moedas
Jogador 1 Venceu!
```

```
Jogo Iniciou!
Para guardar o jogo, no estado atual, inserir 0 0.
Estado do jogo:
Fila 1: 2 moedas
Fila 2: 3 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
2 1
Estado do jogo:
Fila 1: 2 moedas
Fila 2: 2 moedas
A máquina escolheu a fila 1 e retirou 2 moeda(s).
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 2 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
2 2
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 0 moedas
Perdeu!
```

5.2. Resultados obtidos

Os testes mostraram que o programa funciona corretamente em diversos cenários, incluindo validações, jogadas estratégicas e gestão de estados de jogo.

- Comportamento do Menu: todas as opções do menu são exibidas e funcionam como esperado.

- Validação de entradas: entradas inválidas são rejeitadas e mensagens de erro claras são exibidas.
- Jogabilidade: tanto no modo Humano X Humano quanto no modo Humano X Máquina, as jogadas são realizadas corretamente.
- Máquina avançada: demonstrou boa capacidade estratégica, ajustando as suas jogadas conforme as condições do jogo.
- Salvar e continuar jogo: o estado do jogo é salvo e carregado corretamente, com o ficheiro a ser excluído após o jogo carregado acabar.

5.3. Exemplo de interação completa

- Configuração inicial:

```
Por favor digite a opção que pretende:
    1 - Iniciar Novo Jogo
    2 - Continuar Jogo Guardado
    3 - Sair
1
Por favor digite o modo que pretende:
    1 - Humano X Humano
    2 - Humano X Máquina Básica
    3 - Humano X Máquina Avançada
    4 - Voltar ao Menu anterior
1
Quantas filas deseja no seu jogo?
3
Quantas moedas tem a fila 1?
5
Quantas moedas tem a fila 2?
4
Quantas moedas tem a fila 3?
7
```

- Jogadas e finalização:

```
Jogo Iniciou!
Para guardar o jogo, no estado atual, inserir 0 0.
Estado do jogo:
Fila 1: 5 moedas
Fila 2: 4 moedas
Fila 3: 7 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
1 3
Estado do jogo:
Fila 1: 2 moedas
Fila 2: 4 moedas
Fila 3: 7 moedas
Jogador 2 - Escolha a fila de moedas e o número de moedas a retirar.
2 4
Estado do jogo:
Fila 1: 2 moedas
Fila 2: 0 moedas
Fila 3: 7 moedas
Jogador 1 - Escolha a fila de moedas e o número de moedas a retirar.
3 6
Estado do jogo:
Fila 1: 2 moedas
Fila 2: 0 moedas
Fila 3: 1 moedas
Jogador 2 - Escolha a fila de moedas e o número de moedas a retirar.
1 2
Estado do jogo:
Fila 1: 0 moedas
Fila 2: 0 moedas
Fila 3: 1 moedas
Jogador 2 Venceu!
```

Estes testes asseguram que o programa cumpre os requisitos e proporciona uma experiência funcional e interativa para os utilizadores.

6. Conclusão

A realização deste trabalho foi uma oportunidade valiosa para aplicar conceitos fundamentais da matéria lecionada ao longo do semestre, como modularidade, alocação dinâmica de memória, validação de dados e manipulação de ficheiros. Ao longo do desenvolvimento e dos testes, foram enfrentados diversos desafios que contribuíram significativamente para a aprendizagem e aperfeiçoamento das nossas habilidades enquanto programadores.

6.1. Dificuldades enfrentadas

Enquanto grupo, conseguimos utilizar o que cada um dos membros tem de melhor mas, apesar disso, encontrámos as seguintes dificuldades:

- Modularização do código: organizar o programa de forma clara e reutilizável, dividindo-o em funções bem definidas, foi desafiador no início pois não sabíamos por onde começar, até recomeçámos o trabalho por completo algumas vezes. No entanto, a utilização de funções específicas para tarefas como definição de filas, exibição do estado de jogo, entre outras, facilitou a manutenção e compreensão do código.
- Gestão de memória: implementar a alocação e liberação de memória de forma adequada (usando o *malloc* e o *free*) foi uma etapa crítica para evitar fugas de memória. Durante os testes, surgiram problemas relacionados à manipulação incorreta de ponteiros, que foram corrigidos após uma análise detalhada do código.
- Implementação da lógica avançada da máquina: desenvolver a estratégia da máquina no modo avançado, que exige decisões baseadas em cálculos estratégicos, exigiu vários testes e ajustes para alcançar um comportamento que se alinhasse com os objetivos do jogo.

6.2. Lições aprendidas

Este trabalho proporcionou maior experiência prática em programação. Dito isto, podemos afirmar que aprendemos que:

- Validar entradas do utilizador desde o início evita erros inesperados e ajuda a tornar o programa mais robusto.
- Utilizar estruturas é uma técnica eficaz para facilitar a manipulação e a compreensão do código.

- Investir tempo no planeamento da estrutura do programa e realizar testes detalhados para diferentes cenários, garante que o software cumpra aos requisitos especificados no enunciado.
- Colaborar em equipa permite identificar e corrigir erros rapidamente, além de proporcionar troca de conhecimentos entre os membros.

6.3. Reflexões finais

Este projeto não apenas consolidou os fundamentos da programação em C, mas também apresentou desafios práticos que exigiram a aplicação de habilidades de resolução de problemas e pensamento lógico.

A funcionalidade do programa foi verificada aplicando diversos testes, garantindo que ele cumprisse todos os requisitos estabelecidos. Embora tenha havido dificuldades, como a implementação de estratégias avançadas da máquina, o resultado final foi um jogo interativo, funcional e estrategicamente envolvente.

Com este trabalho, adquirimos uma visão mais clara sobre a importância da estruturação de código, da gestão de memória e da validação de entradas, conhecimentos que serão essências para o futuro.